
Proceedings Paper:
Yiangos, G, Melinos, A, Kelly, T orcid.org/0000-0002-6575-3682 et al. (1 more author) (Accepted: 2021) Projective Urban Texturing. In: Proceedings of the International Conference on 3D Vision 2021. 3DV 2021: International Conference on 3D Vision, 01-03 Dec 2021, Online. IEEE . (In Press)

Projective Urban Texturing

Yiangos Georgiou
CYENS

Melinos Averkiou
CYENS

Tom Kelly
University of Leeds

Evangelos Kalogerakis
University of Massachusetts Amherst

Figure 1: Our system learns to transfer the styles of panoramic data sources (inset, left) to "polygon-soup meshes" (blue, right), generating high resolution detailed urban textures (centre).

Abstract

The creation of high quality textures for immersive urban environments is a central component of the city modeling problem. Many recent pipelines capture or synthesize large quantities of city geometry using scanners or procedural modeling pipelines. Such geometry is intricate and realistic, however the generation of photo-realistic textures for such large scenes remains a problem - photo datasets are often panoramic and are challenging to re-target to new geometry. To address these issues we present a neural architecture to generate photo-realistic textures for urban environments. Our Projective Urban Texturing (PUT) system iteratively re-targets textural style and detail from real-world panoramic images to unseen, unstructured urban meshes. The output is a texture atlas, applied onto the input 3D urban model geometry. PUT is conditioned on prior adjacent textures to ensure consistency between consecutively generated textures. We show results for several generated texture atlases, learned from different cities, and present quantitative evaluation of our outputs.

1. Introduction

We address the problem of texture generation for urban geometry. Our goal is to process 3D meshes to create texture maps which are detailed, fit the geometry, and are free from local or global discontinuities. We take a data-driven approach to this problem, using unpaired training over panoramic image datasets to texture cityscapes.

Authoring realistic urban environments is integral to planning, as well as navigating, story-telling, and real-estate. 3D meshes of cities can be reconstructed by numerous 3D acquisition pipelines as well as synthesized by artists or programmers. Such models contain thousands or millions of polygons describing large and small features over urban landscapes, and often lack semantic information. To create vibrant and realistic virtual scenes, these meshes must be textured. Textures provide many of the cues that we use to identify locations and purpose of buildings, including material and color. While some mesh acquisition pipelines provide geometry with materials, frequently they are untextured (if designed by artists), have poor quality textures (e.g. low resolution textures from aerial photogrammetric reconstruction), or we may desire a different textural style. Manually creating high-quality textures is time consuming – not only must the details of the textures match the geometry, but stylistic consistency both within single structures and between adjacent structures or their surroundings (e.g. between buildings, street-furniture, and sidewalks) is also required. An alternative is to programmatically generate textures (e.g. using shape grammars [38]), an approach which can create repeated features (e.g. windows, pillars or lampposts) common in urban landscapes, but is time consuming to write and is often challenged by imperfections or lack of variance in materials (e.g. discoloration and aging).

We present Projective Urban Texturing (PUT), a pipeline to create high quality street-level textures for arbitrary city meshes. Following recent texture generation work, we use a convolutional neural network with adversarial losses to generate textures. However, the resolution of existing approaches is limited by available memory, so they are not able to generate the detail necessary for large environments such as cityscapes – naïvely tiling network outputs creates discontinuities (seams) between adjacent tiles (see Fig-

ure 3). PUT iteratively textures parts of a 3D city mesh by translating panoramic street-level images of real cities using a neural network, and merges the results into a single high-resolution texture atlas. By conditioning the network outputs on previous iterations, we are able to avoid seams and ensure consistent style with prior iterations.

Another central decision when developing texturing systems is the domain to learn from. Learning to create textures in object-space (e.g. over facades) [27, 30] can exploit semantic information to specialise network training and regularise results. However, object-object interactions (e.g. between adjacent buildings or streets and buildings) are not modelled, mesh semantic information is rarely available, and training data is sparse. In contrast, PUT learns in panoramic image-space, texturing multiple objects simultaneously and modeling their interactions consistently.

Panoramic images have become a cheap and fast way to capture noisy texture information about our urban environments [1]. Advances in optics have given us commodity 360 degree panoramic cameras which capture omni-directional information quickly, while efforts such as Google's Street View have captured large datasets of panoramic images for different cities. These images are typically street-level – a single panorama captures multiple objects such opposing facades, overhanging features (e.g. roof eaves), and the street itself. However, due to the real world nature of these images they are often noisy, with occluded buildings (e.g. by trees or vehicles) and camera artefacts (e.g. lens flare). Further, these panoramic image datasets do not contain corresponding 3D geometry. PUT utilises this abundance of panoramic images and unpaired learning to construct a mapping from mesh geometry to urban textures. These are projected onto the geometry to texture large urban areas in a style learnt from the dataset.

PUT offers the following contributions: i) an architecture that learns to generate seamless textures conditioned on the scene meshes and previous results, ii) unpaired style transfer from panoramic image datasets to texture maps in the form of texture atlases, and iii) an iterative pipeline to perform projective texturing over large urban scenes.

2. Related Work

Our work is related to texture generation and blending approaches developed for 3D urban models. Textures are an essential component of modeling urban appearances. They add visual information about geometric and material appearance [18]. Directly projecting photographs onto reconstructed geometry [39, 2, 3] can produce photorealistic results, but requires exact photo locations with consistent geometry (i.e. noise-free location data [31]) and unobstructed views (i.e. no occlusions by trees or vehicles [13]).

Texture synthesis. Traditional learning-based methods generate textures with local and stationary properties [12, 47, 33, 11, 32]. Later work creates larger, more varied textures quickly [10], some of which are domain specific [9]. Procedural languages can create homogeneous textures for urban domains [38], however, these are unrealistic as they lack the heterogeneous appearance of surfaces due to weathering and damage. Such phenomena have been created using simulation or grammars [4, 5, 21, 34]. With the advent of convnets (CNNs), example-based texture synthesis has become a data-driven process, creating textures by gradient descent in feature space [16] or with GANs [52, 15]. Image-to-image translation with a U-Net [43, 27] disentangles the structure from the style to enable style transfer [17]; these have been applied to various domains [19, 30, 51]. Processing panoramic images, such as street-view images, with CNNs has been explored for learning [44] and completion [45]. *Unsupervised* image-to-image networks use novel losses to generate images. For example, a cycle loss [53] learns to translate from one image domain to another, and back again. This technique has been extended to multi-domain [6, 26] and domain-specific problems [20, 46], including faster evaluation with patches [41].

Texturing 3D objects. Creating volumetric 3D details [49, 8] with procedural modeling is a expensive manual process. Cheaper machine learning alternatives are 3D CNNs which can generate voxel colours [40] as well as geometry [50, 48], however these have high memory requirement and there are few available datasets. Attempts to overcome these limitations are ongoing, using a variety of approaches [29, 22, 24, 35, 42, 23]; results are generally low resolution or only in screen-space. GANs have been previously proposed to align and optimize a set of images for a scanned object [25] producing high quality results, however, they require pre-existing captured images for that object. In our case, we leverage large 2D panoramic image datasets for generating textures for urban models without paired image input. A key difference compared to prior work is that our method learns textures by combining contrastive learning-based image synthesis [41] with a novel blending approach to wrap the texture over the geometry.

3. Method

Overview. Given untextured 3D geometry representing an urban environment as input, our pipeline (Figure 2) synthesizes a texture for it. We assume that the surface of the input 3D geometry is unwrapped (parameterized), such that UV coordinates assign a point in a texture atlas (a 2D image map; Figure 2, bottom left) to every point on the geometry. The input urban geometry may come from any source, such as procedural modeling (Figure 1, right) or photogrammetric reconstruction (Figure 7, right).

Our texture synthesis procedure is iterative, working sequentially with selected viewpoints along street centerlines: to begin each iteration, our *rendering module* cre-

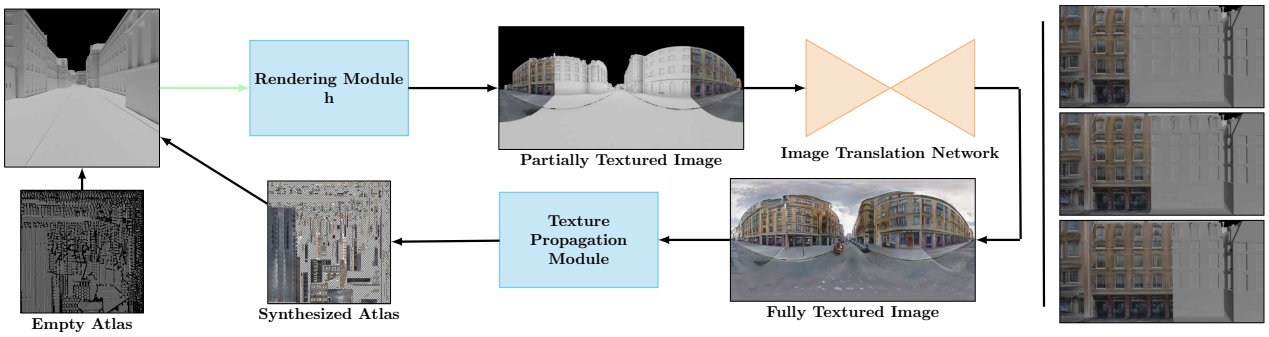

Figure 2: At each iteration our system creates a *partially textured image* by rendering the 3D geometry(top left) using the *synthesized texture atlas* from previous iterations. The *texture propagation module* uses the generated image to update the texture atlas. A visualization of our iterative texturing synthesis output is displayed on a building facade on the right.

ates a panoramic image representing the input 3D geometry from a viewpoint, including any previously generated texture (Figure 2). The rendered images are *partially textured*, since some parts of the 3D urban models may have associated texture synthesized from previous iterations, while others may not contain any texture at all. Untextured parts are rendered in grayscale, and textured parts in colour.

As each iteration continues, the partially textured image is translated into a fully textured RGB image through a neural network (Figure 2, *image translation* network). Our experiments found that passing the partially textured images into the network offered better performance, compared to passing completely untextured (grayscale) images, or using separate images, or channels, for the textured and untextured parts. The network is trained on a set of RGB panorama images taken from streets of a particular city (e.g., London, New York) and a set of rendered images of urban geometry meshes. The two sets are unpaired; we do not assume that the urban geometry meshes have corresponding, or reference, textures. Such a level of supervision would require enormous manual effort to create training textures for each 3D urban model. Our system is instead trained to automatically translate the domain, or geographic "style", of the panorama images of real-world buildings into textures that match the geometry "structure" of the input 3D urban models, and any prior textured parts.

At the end of the iteration, our *texture propagation module* updates the texture atlas from the output fully textured image from the network (Figure 2, *synthesized atlas*). After the final iteration, all viewpoints have been processed, the texture atlas is complete, and is used to render the urban environment, as shown in Figure 5.

Viewpoint selection. In a pre-processing step, we create paths along which we place viewpoints to sequentially capture the input geometry of the city blocks. The paths are formed by following the centerlines of all streets contained in the urban 3D scene (see Section 4). The paths trace the street from the start to the end, "sweeping" the buildings in each block on both sides of the road. Each viewpoint is placed at 5m horizontal intervals along the street centerlines and a vertical height of 2.5m. The horizontal intervals are empirically selected such that each rendered image has an overlap (about 25%) with the image rendered from the previous and next viewpoint. The height selection is motivated by the fact that real-world car mounted cameras are placed at a height close to 2.5m, reducing the "domain gap" between our renderings and real-world panoramas taken from cars. The viewpoint (camera) up and forward axes are set such that they are aligned with the world upright axis and street direction respectively. As a result, the cameras "look" towards the buildings along both sides of the road.

Rendering. Given each viewpoint, each rendering is produced through equirectangular (panoramic) projection. As a result, the domain gap with real-world panoramas is decreased with respect to the projection type. The geometry is rendered with global illumination using Blender [7] at a 512x256 resolution. Any textured parts from previous passes incorporate RGB color information from the texture atlas, while for untextured parts, a white material is used, resulting in a grayscale appearance. Background (e.g., sky) is rendered as black. An example of the resulting "partially textured" image is shown in Figure 2.

Neural network. The input to our image translation network is a partially textured rendering (3x512x256) at each iteration. The neural network uses the architecture from Johnson et al. [28], also used in Contrastive Unpaired Translation (CUT) [41]. The network contains three convolutions, 9 residual blocks, two fractionally-strided convolutions with stride $1/2$, and one convolution layer that maps the features to a RGB image (3x512x256). Since our input is a panoramic image, we use equirectangular convolution for the first convolutional layer of the network [14]. Equirectangular convolution is better suited for processing panoramic images and offers better performance in our experiments. As discussed in Section 4, we follow a training procedure and loss inspired by [41], yet with important differences to handle partially textured images and ensure consistency in the texture generated from different viewpoints.

Texture propagation. At each iteration, we use the generated image from the above network to update the texture atlas. For each texel t with center coordinates (u_t, v_t) in the atlas, we can find the corresponding 3D point $\mathbf{p}_t$ on the input geometry. Then for this 3D point, we find its corresponding pixel location in the generated image at the current iteration i: $[x_{t,i},\ y_{t,i}] = \Pi_i(\mathbf{p_t})$, where Π_i is the equirectangular projection function used during the frame rendering of the current iteration. The color of the texel can be transferred with the mapping: $color[u_t, v_t] = \mathbf{R}_i[x_{t,i},\ y_{t,i}]$, where $\mathbf{R}_i$ is the generated image from the network at the current iteration. However, this strategy can create artifacts since it will overwrite the color of texels that were updated from previous iterations resulting in sharp discontinuities in the generated texture (Figure 3, left).

Figure 3: Texture w/o blending (left), w/ blending (right)

Instead, we follow a pooling strategy, where colors for each texel are aggregated, or blended from different iterations. Specifically, at each iteration, the color of each texel is determined as a weighted average of pixel colors originating from images generated from previous iterations:

$$color[u_t, v_t] = \sum_{i \in V_t} w_{i,t} \mathbf{R}_i[x_{t,i},\ y_{t,i}] \qquad (1)$$

where V_t is the set of iterations where the texel's corresponding 3D point $\mathbf{p}_t$ was accessible (visible) from the cameras associated with these iterations. The blending weights $w_{i,t}$ are determined by how close $\mathbf{p}_t$ was to the center of the projection at each iteration. The closer to the centre the point $\mathbf{p}_t$ was, the higher the weight was set for the color of its corresponding pixel at the generated image. Specifically, if $d_{i,t}$ is the distance of the point $\mathbf{p}_t$ to the projection centerline for the camera at iteration i, the weight of its corresponding pixel color from the generated image at this iteration was determined as $w_{i,t} = 1 - d_{i,t} / \sum_{i'} d_{i',t}$, where i' are iteration indices where the point was visible. The weights were further clamped to $[0.3, 0.7]$ to eliminate contributions from cameras located too far away from the point. Finally, the weights were re-normalized to sum to one. At the final iteration, the texture atlas will be the result of the aggregation from all iterations.

4. Training

To train the image translation neural network of our pipeline, we require a set of training examples from both domains: rendered panorama images from the (untextured) 3D geometry, and real-world panoramic photographs. The two sets are unpaired i.e., the real-world images have no geometric or semantic correspondences with rendered panoramas. We discuss the datasets used for these domains in the following paragraphs, then explain our training procedure.

Real-world panorama dataset. For real-world panoramas, we use the dataset from [36], with $29,414$ images from *London* and $6,300$ images from *New York*; these photos are taken from city streets with vehicle mounted cameras.

Rendered panoramas from 3D geometry. We demonstrate our system on two classes of 3D urban meshes - procedural and photogrammetrically reconstructed meshes Procedural meshes are generated using the CGA language [37]; each generated scene has unique street graph, block-subdivision, and building parameters (e.g. style, feature size, height etc.). We found that the inclusion of details such as pedestrians, trees, and cars helped the multi-layer patch-wise contrastive loss used in our network to identify meaningful correspondences between the real and rendered panorama domains. We also export the street center-line data from the procedural models (no other "semantic" information or labels are used). We generated 10 scenes with 1600 viewpoints sampled from 18 random paths on streets of each scene. This resulted in $16K$ viewpoints in total, from which we rendered grayscale images of the input geometry using equirectangular projection. We use 9 scenes ($14.4K$ grayscale, rendered panoramic images) for training, and keep 1 scene for testing ($1.6K$ images). We note that the streets, building arrangements and their geometry were unique for each street and scene, thus, the test synthetic renderings were different from the training ones.

The photogrammetric mesh dataset is from Google Earth. The meshes are coarse and noisy (Fig. 7, right, blue mesh). They contain holes, vehicles, and trees. This contrasts the precise, high-detail, data in procedural meshes. Street centerlines are collected from OpenStreetMap. We use 2 scenes and 1411 rendered images. One is used for training and the other for testing.

Training Procedure. As mentioned above, since there are no target reference texture images for the input 3D geometry, we resort to a training procedure that uses the unpaired real-world RGB panoramas and our partially textured panorama renderings. Contrastive Unpaired Translation (CUT) [41] is a fast and flexible method to train image translation networks between unpaired domains. One potential strategy is to use CUT's losses and training as-is in our setting. However, this strategy disregards the need to ensure consistency between the generated textures at different iterations of our pipeline. In addition, our image translation network processes partially textured images, which are different from both the domain of real-world panoramas and the domain of grayscale rendered images. Next,

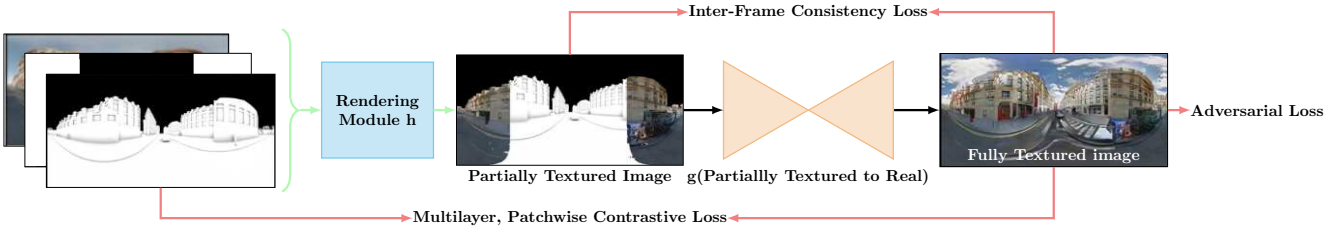

Figure 4: Training procedure: the green arrow shows the mapping h from a grayscale rendering of a training city block to the partially textured image. This mapping is performed in our rendering module that integrates previous texture information with the help of a binary mask indicating prior textured regions. The image translation network g maps the partially textured image to a fully textured one, mimicking real-world statistics with the help of an adversarial loss. The partially-textured image is compared to the fully textured image in the inter-frame consistency loss to discourage seams in the output texture.

we explain how our training procedure handles the different kinds of inputs required for our image translation network and also promotes consistency between outputs of different iterations.

Multi-layer Patch-wise Contrastive Loss. The goal of our training procedure is to learn the parameters of our image translation network $g : P \rightarrow R$ mapping the domain P of partially textured images to the domain R of fully textured ones mimicking real-world statistics. To associate input and output structure we use a multi-layer patch-wise contrastive loss between input and output. The association between images is achieved by the comparison of the stack of features for selected layers in the encoder network f.

Specifically, in our setting, given an untextured (grayscale) rendering S from our set of untextured renderings $\mathbf{S}$, we first convert it to a partially textured one by integrating previously textured parts indicated from a binary mask M (1 for previously textured parts, 0 for untextured ones). This conversion $h : S \rightarrow P$ is a fixed, parameter-free mapping implemented in our rendering module. Given a rendered image S and the generated image $g(h(S))$, we generate their stack of features $z = f(S)$ and $\hat{z} = f(g(h(S)))$. Pairs of patches from the rendered and generated image at the same location n are treated as positive, while pairs from different locations are treated as negative. A temperature-scaled cross-entropy loss H (with temperature $\tau = 0.07$) computes the probability of the positive example being selected over negatives [41]. The loss is summed over our dataset and samples:

$$\mathcal{L}_{contr_S} = \mathbb{E}_{S\sim\mathbf{S}} \sum_{l=1}^{L} \sum_{n=1}^{N_l} H(\hat{z_l^n}, z_l^n, z_l^{N_l/n}) \qquad (2)$$

where L is the number of selected layers and N_l is the index set of spatial locations in feature maps at each layer. As in CUT [41], to avoid unnecessary generator changes, a cross-entropy loss $\mathcal{L}_{contr_R}$ is also applied on patches sampled from the real images domain $\mathcal{R}$.

Adversarial losses. Apart from the contrastive losses, we make use of adversarial loss to ensure that the fully textured images generated from the image translation network g share similar statistics with the real-world panoramas:

$$\mathcal{L}_{gan} = \mathbb{E}_{R\sim\mathbf{R}}[log(d_r(R))] + \mathbb{E}_{S\sim\mathbf{S}}[1 - log(d_r(g(h(S))))]$$

where $\mathbf{R}$ is the training set of real-world panoramas, d_r is a discriminator network following the architecture of CUT applied to real images domain.

Inter-frame consistency loss. We introduce an inter-frame consistency loss to promote consistency in the generated textured images from our image translation network g across different iterations of our pipeline loop. Given a fully textured image $g(h(S))$, generated from the image translation network during training, and a partially textured image $h(S)$ containing texture from previous iterations, we compare the textured regions in $h(S)$ with the corresponding generated regions. The comparison is performed with the help of the binary mask M which contains ones for the partially textured regions of $h(S)$ and zeros for the rest. Using the mappings defined above, the loss is expressed as follows:

$$\mathcal{L}_{cons} = \mathbb{E}_{S\sim\mathbf{S}}[\| (g(h(S))) \odot M - h(S) \odot M \|_1] \qquad (3)$$

where $\odot$ is the Hadamard product. We discuss the effect of this loss in our experiments and ablation study.

Full Objective. The full objective is a weighted combination of the above losses:

$$\mathcal{L} = \lambda_1\mathcal{L}_{gan} + \lambda_2\mathcal{L}_{cons} + \lambda_3\mathcal{L}_{contr_S} + \lambda_4\mathcal{L}_{contr_R} \qquad (4)$$

where $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ are hyper-parameters set to $1.0, 10.0, 0.5, 0.5$ respectively.

Implementation details. We use the Adam optimizer with learning rate $2 \cdot 10^{-4}$ to train the above architecture. We train the model for 200 epochs. During the first 150 epochs the learning rate is fixed, then it gradually decays for the last 50 epochs. All our code and datasets, including the rule-set and scripts for the procedural generation of the training and test scenes will be made available online (upon acceptance) at http://github.com/xxx/PUT.

Figure 5: Renderings of our test scene textured by PUT trained on London panoramas. We show street-level renderings.

5. Evaluation

We now evaluate our pipeline on test models generated with the process described in Section 4. As discussed earlier, no identical city blocks exist in our train and test meshes. We trained our translation network separately on the dataset of *London* and the dataset of *New York* images.

Qualitative Evaluation. First, we evaluate the quality of our method's output textures by placing perspective cameras at the street level for each of our test meshes and rendering the meshes with texture atlases created by our method. Results can be seen in Figure 5, for the network trained on London data. Our method transfers the appearance of large structures such as multi-storey walls, streets, or sidewalks from real panoramas and often aligns texture details with geometric ones i.e., texture edges of windows and doors with geometry edges. A side-effect of our method is that real-world shadows can also be translated to streets and sidewalks (Figure 5).

City style transfer. Our method can be trained on street-level panoramic images collected from different cities in the world to give urban 3D models the appearance of a city's distinctive style, as illustrated by our London and New York examples in Figure 6. The generated textured city blocks appear distinctively textured with our pipeline trained on New York versus the ones textured with London panoramas. For example, a red brick-like appearance is visible in the walls of the meshes textured by our network trained in New York panoramas; such appearance is common in New York buildings. In contrast, meshes textured by our network trained on London panoramas appear with white, yellow and brown distributions of color on their facades, a color distribution which is common around London area.

Mesh type generalization. We also demonstrate our pipeline on an unstructured city 3D model from Google Earth, shown in Figure 7. Our method generates consistent facade textures, even for such very challenging polygon soups that do not contain any facade or window geometric information as demonstrated in the untextured rendering of the same model (Figure 7, right). However, the absence of accurate facade geometry limits the diversity of texturing.

Inter-frame consistency. We qualitatively evaluate the degree of consistency between consecutive output textures from our architecture in Figure 8. We display six consecutive intermediate panoramic outputs of our image translation network trained on the London dataset. We note that

Figure 6: Street-level renderings of buildings in our test scene textured by our method trained on London images (left) and New York images (right).

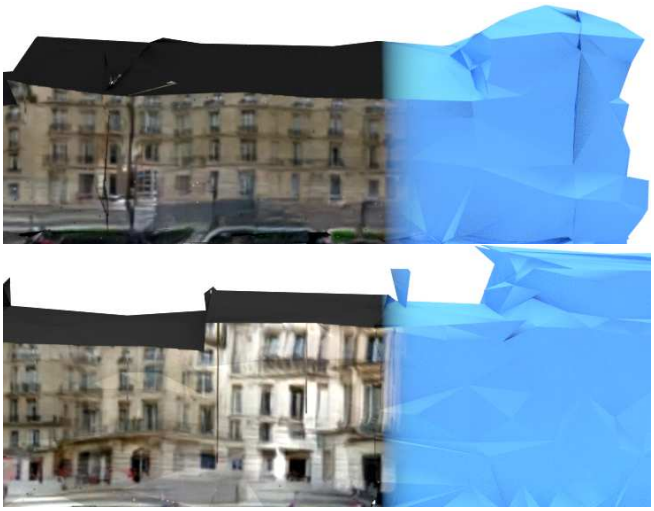

Figure 7: Our method can generate textures for challenging city polygon soups such as those found in Google Earth. Here we show a rendering (at street-level) of buildings textured by our method trained on London panoramas, for a Google Earth mesh. Notice how the untextured part of the buildings (right) reveals non-planar walls and absence of any window or door cues, yet our method manages to create a plausible texture for the facades.

each consecutive output is consistent with the preceding images. This helps to preserve details in the generated texture maps, as seen at the top of Figure 8 which shows a close-up rendering of textured 3D mesh on the right side of the street.

Figure 8: A sequence of six consecutive panoramic outputs of our network trained on London data (bottom). Notice how each consecutive output is consistent with the previous, which allows our texture propagation module to preserve details and accurately align features like windows and doors on facade geometry (top).

Figure 9: Tree artifacts appear on the facades in images generated from *CUT* (top row). Our inter-frame consistency loss prevents these artifacts (bottom row).

Figure 10: Style discontinuities occur between consecutive images generated from *CUT* (top row) in contrast with PUT which generates consistent colors between adjacent viewpoints (bottom row).

Comparison with *CUT*. We compared PUT to the *CUT* network for generating the texture atlas. We note that *CUT* does not condition its image translation network on the partially textured image and does not use our inter-frame consistency loss. Both networks were trained on the London dataset. For fair comparison, we the same equirectangular convolution for the first convolution layer of both architectures. Table 1 shows FID scores for panoramic renderings

produced by PUT and *CUT* for our test procedural scene. Since a large area of panoramic images consists of road and sky with little or no variation in texture, we also evaluate the FID on cropped versions of these renderings that isolate the facades from the roads and sky; we call this variant score as "crop FID". We additionally show the performance of PUT and *CUT* using three different blending approaches: *no blend* where no blending is applied between frames, *average* blending which takes the average RGB values between frames and our texture propagation (without weights), *weighted* uses the weighted averaging scheme of Equation 1. Our model outperforms *CUT* regardless of the blending approach. Furthermore, our texture propagation approach based on our weighted averaging scheme outperforms the no-blend or average-blend baselines.

In Figures 9 and 10 we show qualitative comparisons between PUT and *CUT*. Figure 9 (top) shows failure cases of *CUT*, i.e. undesired tree artifacts on the facades which PUT decreases (bottom) by using the inter-frame consistency loss. Moreover, the style of the facade between the first and the third panorama produced by *CUT* differs in Figure 10 (top) in contrast to PUT which generates more consistent results (bottom).

	No-Blend↓		**Average↓**		**Weighted↓**	
-	full	crop	full	crop	full	crop
CUT	131.56	110.23	135.16	113.39	121.65	101.12
PUT	128.19	95.60	132.59	97.25	**115.38**	**86.30**

Table 1: FID comparison between PUT and *CUT*

Ablation study. We also designed an ablation study to support the three main design choices for our network architecture, namely the use of the mask M in the inter-frame consistency loss (Equation 3), the use of equirectangular convolution for the first convolutional layer of the network [14], and the merging of the untextured (grayscale) and partially textured image to create 3-channel inputs.

Table 2 reports the "crop FID" scores for five variations of PUT produced by combination of the three design choices we made. Each row is a different model, incorporating different variations of the three choices. Comparing PUT 1 and PUT 2 we can see that our choice to merge untextured and partially textured images in a 3-channel input improves FID scores. Adding equirectangular convolution in the first convolution layer of PUT 3 further slightly improves FID score, while incorporating the mask in the inter-frame consistency in PUT 4 marks a much larger improvement compared to PUT 2. Finally, adding all three, results in the best FID score for the full model.

Model	**Masked Cons.**	**Equir. Conv.**	**Gray+RGB**	**crop FID↓**
PUT 1	–	–	–	93.10
PUT 2	–	–	✓	92.47
PUT 3	–	✓	✓	92.23
PUT 4	✓	–	✓	87.56
full PUT	✓	✓	✓	**86.30**

Table 2: FID scores for PUT design choices. "Masked Cons." means whether we use the mask M in the consistency loss, "Equir. Conv." means whether we use equirectangular convolution, "Gray+RGB" means whether we merge the grayscale rendering with the RGB partially textured image in a 3-channel image (or treat them as 4-channel image).

6. Limitations and Conclusion

We have presented PUT, a method for texturing urban 3D scenes. Our method is able to texture large 3D meshes of urban areas using street-view panoramic images in an unpaired manner without requiring any correspondences between the 3D mesh and the real-world street-view panoramas. We demonstrate that our method is able to generate realistic and consistent textures for large areas.

One limitation of our method is its inability to texture roofs using street-view panoramas; we hope to resolve this issue in future work by incorporating aerial images into our pipeline. Another limitation of PUT is that, compared to city texturing approaches which tile and repeat the same texture over hundreds of locations, our single large texture atlas is memory intensive. However, recent advances in rendering pipelines have made it possible handle such large textures, while consumer-level GPUs offer ever increasing amounts of dedicated memory. Our data-driven approach can be challenged by the large domain-gaps in the distributions of features between panoramic photos and geometry, such as large red buses in London or trees in New York images. A related issue is the reconstruction of artifacts and noise present in the panorama datasets – for example cars, lens aberrations, and shadows over the building – while some of these features (such as shadows) can be useful, the lack of control; such issues are common in data-driven systems.

In the future, we would like to allow users to explore the style-space of an urban environment automatically by navigating a latent space in the generative networks. This would give users an ability to create unique, new styles which are different from the available training data, or travel through time. Finally, we imagine an extension to PUT which would further increase our texture resolution to better portray fine details in our textures using super-resolution neural architectures. In particular, we would like to vary the resolution based on the viewpoint, to generate millimeter resolution textures close to the camera, and consistent cityscapes in the far distance, in real-time.

References

[1] Dragomir Anguelov, Carole Dulong, Daniel Filip, Christian Frueh, Stéphane Lafon, Richard Lyon, Abhijit Ogale, Luc Vincent, and Josh Weaver. Google street view: Capturing the world at street level. *Computer*, 43(6):32–38, 2010. 2

[2] Jan Böhm. Multi-image fusion for occlusion-free façade texturing. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 35(5):867–872, 2004. 2

[3] Alexander Bornik, Konrad Karner, Joachim Bauer, Franz Leberl, and Heinz Mayer. High-quality texture reconstruction from multiple views. *The journal of visualization and computer animation*, 12(5):263–276, 2001. 2

[4] Yao-Xun Chang and Zen-Chung Shih. The synthesis of rust in seawater. *The Visual Computer*, 19(1):50–66, 2003. 2

[5] Yanyun Chen, Lin Xia, Tien-Tsin Wong, Xin Tong, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Visual simulation of weathering by γ-ton tracing. In *ACM SIGGRAPH*, pages 1127–1133. 2005. 2

[6] Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8789–8797, 2018. 2

[7] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. 3

[8] Barbara Cutler, Julie Dorsey, Leonard McMillan, Matthias Müller, and Robert Jagnow. A procedural approach to authoring solid models. *ACM Trans. Graph.*, 21(3):302–311, July 2002. 2

[9] Dengxin Dai, Hayko Riemenschneider, Gerhard Schmitt, and Luc Van Gool. Example-based facade texture synthesis. pages 1065–1072, 2013. 2

[10] Soheil Darabi, Eli Shechtman, Connelly Barnes, Dan B Goldman, and Pradeep Sen. Image melding: combining inconsistent images using patch-based synthesis. *ACM SIGGRAPH*, 31(4):1–10, 2012. 2

[11] Alexei A Efros and William T Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 341–346, 2001. 2

[12] Alexei A Efros and Thomas K Leung. Texture synthesis by non-parametric sampling. In *IEEE ICCV*, volume 2, pages 1033–1038. IEEE, 1999. 2

[13] Lubin Fan, Przemyslaw Musialski, Ligang Liu, and Peter Wonka. Structure completion for facade layouts. *ACM SIGGRAPH Asia*, 33(6), Nov. 2014. 2

[14] Clara Fernandez-Labrador, Jose M. Facil, Alejandro Perez-Yus, Cendric Demonceaux, Javier Civera, and Jose J. Guerrero. Corners for layout: End-to-end layout recovery from 360 images. 2019. 3, 8

[15] Anna Frühstück, Ibraheem Alhashim, and Peter Wonka. Tilegan: synthesis of large-scale non-homogeneous textures. *ACM SIGGRAPH*, 38(4):1–11, 2019. 2

[16] Leon Gatys, Alexander S Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. In *NIPS*, pages 262–270, 2015. 2

[17] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016. 2

[18] Michael Gruber, Marko Pasko, and Franz Leberl. Geometric versus texture detail in 3-d models of real world buildings. In *Automatic extraction of Man-made objects from aerial and space images*, pages 189–198. Springer, 1995. 2

[19] Éric Guérin, Julie Digne, Éric Galin, Adrien Peytavie, Christian Wolf, Bedrich Benes, and Benoît Martinez. Interactive example-based terrain authoring with conditional generative adversarial networks. *ACM TOG*, 36(6):1–13, 2017. 2

[20] Xi Guo, Zhicheng Wang, Qin Yang, Weifeng Lv, Xianglong Liu, Qiong Wu, and Jian Huang. Gan-based virtual-to-real image translation for urban scene semantic segmentation. *Neurocomputing*, 2019. 2

[21] Tobias Günther, Kai Rohmer, and Thorsten Grosch. GPU-accelerated Interactive Material Aging. In Michael Goesele, Thorsten Grosch, Holger Theisel, Klaus Toennies, and Bernhard Preim, editors, *Vision, Modeling and Visualization*. The Eurographics Association, 2012. 2

[22] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. *ACM TOG*, 37(6):1–15, 2018. 2

[23] Paul Henderson, Vagia Tsiminaki, and Christoph H Lampert. Leveraging 2d data to learn textured 3d mesh generation. pages 7498–7507, 2020. 2

[24] Philipp Henzler, Niloy J Mitra, , and Tobias Ritschel. Learning a neural 3d texture space from 2d exemplars. In *IEEE CVPR*, June 2019. 2

[25] Jingwei Huang, Justus Thies, Angela Dai, Abhijit Kundu, Chiyu Max Jiang, Leonidas Guibas, Matthias Nießner, and Thomas Funkhouser. Adversarial texture optimization from rgb-d scans. *arXiv preprint arXiv:2003.08400*, 2020. 2

[26] Xun Huang, Ming-Yu Liu, Serge Belongie, and Jan Kautz. Multimodal unsupervised image-to-image translation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 172–189, 2018. 2

[27] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *IEEE CVPR*, 2017. 2

[28] Justin Johnson, Alexandre Alahi, and Fei fei Li. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, 2016. 3

[29] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3907–3916, 2018. 2

[30] Tom Kelly, Paul Guerrero, Anthony Steed, Peter Wonka, and Niloy J Mitra. Frankengan: guided detail synthesis for building mass models using style-synchonized gans. *ACM TOG*, 37(6):1–14, 2018. 2

[31] Bryan Klingner, David Martin, and James Roseborough. Street view motion-from-structure-from-motion. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 953–960, 2013. 2

[32] Chuan Li and Michael Wand. Approximate translational building blocks for image decomposition and synthesis. *ACM Transactions on Graphics (TOG)*, 34(5):1–16, 2015. 2

[33] Lin Liang, Ce Liu, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. Real-time texture synthesis by patch-based sampling. *ACM Trans. Graph.*, 20(3):127–150, July 2001. 2

[34] Jianye Lu, Athinodoros S Georghiades, Andreas Glaser, Hongzhi Wu, Li-Yi Wei, Baining Guo, Julie Dorsey, and Holly Rushmeier. Context-aware textures. *ACM Transactions on Graphics (TOG)*, 26(1):3–es, 2007. 2

[35] Moustafa Meshry, Dan B Goldman, Sameh Khamis, Hugues Hoppe, Rohit Pandey, Noah Snavely, and Ricardo Martin-Brualla. Neural rerendering in the wild. In *IEEE CVPR*, pages 6878–6887, 2019. 2

[36] Piotr Mirowski, Andras Banki-Horvath, Keith Anderson, Denis Teplyashin, Karl Moritz Hermann, Mateusz Malinowski, Matthew Koichi Grimes, Karen Simonyan, Koray Kavukcuoglu, Andrew Zisserman, et al. The streetlearn environment and dataset. *arXiv preprint arXiv:1903.01292*, 2019. 4

[37] Pascal Mueller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. Procedural modeling of buildings. *ACM TOG*, 25(3):614–623, 2006. 4

[38] Pascal Müller, Gang Zeng, Peter Wonka, and Luc Van Gool. Image-based procedural modeling of facades. *ACM SIGGRAPH*, 26(3):85, 2007. 1, 2

[39] Wolfgang Niem and Hellward Broszio. Mapping texture from multiple camera views onto 3d-object models for computer animation. In *Proceedings of the International Workshop on Stereoscopic and Three Dimensional Imaging*, pages 99–105, 1995. 2

[40] Michael Oechsle, Lars Mescheder, Michael Niemeyer, Thilo Strauss, and Andreas Geiger. Texture fields: Learning texture representations in function space. In *IEEE ICCV*, pages 4531–4540, 2019. 2

[41] Taesung Park, Richarf Zhang Alexei A. Efros, and Jun-Yan Zhu. Contrastive learning for unpaired image-to-image translation. *ECCV*, 2020. 2, 3, 4, 5

[42] Amit Raj, Cusuh Ham, Connelly Barnes, Vladimir Kim, Jingwan Lu, and James Hays. Learning to generate textures on 3d meshes. In *IEEE CVPR Workshops*, June 2019. 2

[43] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 2

[44] Alisha Sharma and Jonathan Ventura. Unsupervised learning of depth and ego-motion from cylindrical panoramic video. *arXiv preprint arXiv:1901.00979*, 2019. 2

[45] Shuran Song, Andy Zeng, Angel X Chang, Manolis Savva, Silvio Savarese, and Thomas Funkhouser. Im2pano3d: Extrapolating 360 structure and semantics beyond the field of view. In *IEEE CVPR*, pages 3847–3856, 2018. 2

[46] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. Video-to-video synthesis. *arXiv preprint arXiv:1808.06601*, 2018. 2

[47] Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, page 479–488, USA, 2000. ACM Press/Addison-Wesley Publishing Co. 2

[48] Chao Wen, Yinda Zhang, Zhuwen Li, and Yanwei Fu. Pixel2mesh++: Multi-view 3d mesh generation via deformation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1042–1051, 2019. 2

[49] Emily Whiting, John Ochsendorf, and Frédo Durand. Procedural modeling of structurally-sound masonry buildings. *ACM SIGGRAPH*, 28(5):1–9, Dec. 2009. 2

[50] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *NIPS*, pages 82–90, 2016. 2

[51] X Zhang, C May, and D Aliaga. Synthesis and completion of facades from satellite imagery. 2020. 2

[52] Yang Zhou, Zhen Zhu, Xiang Bai, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. Non-stationary texture synthesis by adversarial expansion. *ACM SIGGRAPH*, 37(4), July 2018. 2

[53] Zhu, Jun-Yan, Park Taesung, Isola, Phillip, and Efros Alexei A. Unpaired image-to-image translation using cycle-consistent adversarial networks. *IEEE ICCV*, 2017. 2